Web Component

L'interopérabilité en web

Expliqué par David Gilson

https://github.com/gilsdav

C’est quoi ?

- **Wikipédia:**

  « **Web Components** are a set of features that provide a standard component model for the Web[1] allowing for encapsulation and interoperability of individual HTML elements. »

- Basé sur quartes technologies
  - Custom Elements
  - HTML Templates
  - Shadow DOM
  - ES Modules

ES Modules

- API ES6 permetant l'export de class, fonction ou tout autre type de variable

```
<script type="module">
    export class Animal {
        constructor(name, species) {
            this.name = name;
            this.species = species;
        }

        greet() {
            return `Hello, my name is ${this.name}!`;
        }
    }

    export function greet(name){
        return `Hello, my name is ${name}!`;
    }
</script>
```

Custom Elements

- Ensemble d'API permettant d'étendre un élément HTML, d'un créer un nouveau et de définir leur comportement.

```
customElements.define('word-count', WordCount, { extends: 'p' });
```

- Création d'un élément « word-count » avec la logique de la classe « WordCount » et qui étend « p ».

```
class WordCount extends HTMLParagraphElement {
    constructor() {
        super();
    }
}
```

Custom Elements

- Création d'un composant autonome

```
<script>
    class PopUpInfo extends HTMLElement {
        constructor() {
            super();
        }

        connectedCallback() {
            const text = this.getAttribute('text');
            this.innerHTML = '<span>' + text + '</span>' ;
        }
    }

    customElements.define('popup-info', PopUpInfo);
</script>

<popup-info text="Hey there"></popup-info>
```

Custom Elements

- Cycle de vie:
  - connectedCallback()
    - Ajouté dans le document
  - disconnectedCallback()
    - Supprimé du document
  - attributeChangedCallback(name, oldValue, newValue)
    - Quand un attribut listé dans « observedAttributes » change
  - static get observedAttributes()

HTML Templates

- Partie de code HTML qui n'est pas rendu (pas visible) mais qui est clonable afin de pouvoir être inséré dans le DOM au runtime.
- Ce « fragment » n'est parsé qu'une seule fois (optimisation des performances comparé a du copié collé dans l'HTML).

HTML Templates

```
<template id="myTemplate">
    <h1>The ultimate guide to Web Components!</h1>
    <p>What do you want to learn about Web Components today?</p>
</template>

<script>
    const template = document.querySelector('#myTemplate’);
    const clone = document.importNode(template.content, true);
    document.body.appendChild(clone);
</script>
```

Shadow DOM

- Encapsulation de style et de balises dans une sous-arborescence DOM.
  - Shadow Tree = la sous arborescence
  - Shadow Host = élément auquel la sous arborescence est liée
- Garanti qu'un composant fonctionnera peu importe l'environement aussi bien au niveau CSS que JavaScript.

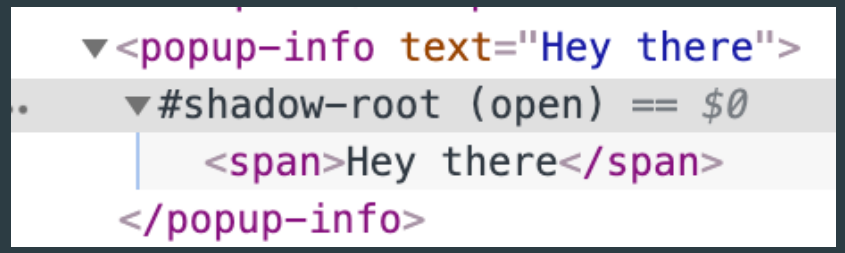
Shadow DOM

```
const shadow = element.attachShadow({mode: 'open'});
```

Shadow DOM

```
<script>
     class PopUpInfo extends HTMLElement {
          wrapper;
          constructor() {
               super();

               // Création de l'élément shadow root
               const shadow = this.attachShadow({mode: 'open'});

               this.wrapper = document.createElement('span');

               shadow.appendChild(this.wrapper);

          }

          connectedCallback() {
               const text = this.getAttribute('text');
               this.wrapper.innerHTML = text;
          }
     }

     customElements.define('popup-info', PopUpInfo);
</script>
```

Shadow DOM


<popup-info text="Hey there">
#shadow-root (open) == $0
Hey there
</popup-info>

Quelques règles

- L’enregistrement d’un Custom Element ne peut se faire **qu’une seule fois**
- Un Web Component ne peu pas être **auto fermant**
- Il est obligatoire d’avoir au moins un « **-** » dans le **nom**
  - <mycard></mycard> ou <CardComponent></CardComponent> sont **invalide**
  - <my-card></my-card> est **valide**

Web Component

```
class AwesomeCardComponent extends HTMLElement {
     titleElement;

     constructor() {
          super();

          const shadow = this.attachShadow({mode: 'open'});
          const template = document.querySelector('#awesomeCardComponentTemplate');
          const wrapper = document.importNode(template.content, true);
          this.titleElement = wrapper.querySelector('h1'); // Uniquement possible ici
          shadow.appendChild(wrapper);

          const style = document.createElement('style');
          shadow.appendChild(style);
     }

     get greeting() {
          return this.getAttribute('greeting');
     }
     set greeting(val) {
          if (val) {
               this.setAttribute('greeting', val);
          } else {
               this.removeAttribute('greeting');
          }
     }

     ...
}
```

Web Component

```
class AwesomeCardComponent extends HTMLElement {
    ...

    static get observedAttributes() {
        return ['greeting'];
    }

    attributeChangedCallback(name, oldValue, newValue) {
        if (name === 'greeting' && oldValue !== newValue) {
            this.titleElement.innerHTML = newValue;
        }
    }

}
```

```
customElements.define('awesome-card', AwesomeCardComponent);
```

```
<awesome-card greeting="Hello world!"></awesome-card>
```

Web Component

```
this.dispatchEvent(new CustomEvent('onRemove', { detail: this.index }));
```

```
const card = document.createElement('awesome-card ');
card.addEventListener('onRemove', this._removeTodo.bind(this));
```

Angular Elements

Angular Elements

```
ng add @angular/elements --name=*your_project_name*
```

```
npm install @webcomponents/webcomponentsjs --save
```

```
// Polyfills
import '@webcomponents/webcomponentsjs/custom-elements-es5-adapter.js';
```

Angular Elements

```
const PopupElement = createCustomElement(PopupComponent, {injector});

customElements.define('popup-element', PopupElement);
```

```
document.body.appendChild(document.createElement('popup-element'));
```